42P17501

UNITED STATES PATENT APPLICATION

FOR

A PERIPHERAL DEVICE HAVING A PROGRAMMABLE IDENTIFICATION CONFIGURATION REGISTER

INVENTOR:

PAUL A. JACOBS

ERIC J. DEHAEMER

PREPARED BY:

BLAKELY, SOKOLOFF, TAYLOR & ZAFMAN
12400 WILSHIRE BOULEVARD
SEVENTH FLOOR
LOS ANGELES, CA 90025-1026

(408) 720-8300

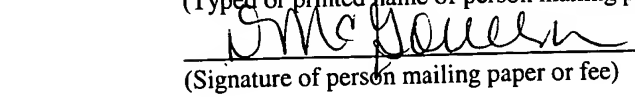
EXPRESS MAIL CERTIFICATE OF MAILING

"Express Mail" mailing label number: EV336584868US

Date of Deposit: September 23, 2003

I hereby certify that I am causing this paper or fee to be deposited with the United States Postal Service "Express Mail Post Office to Addressee" service on the date indicated above and that this paper or fee has been addressed to the Commissioner for Patents, PO Box 1450, Alexandria, Virginia 22313-1450

Deborah A. McGovern
(Typed or printed name of person mailing paper or fee)



(Signature of person mailing paper or fee)

September 23, 2003
(Date signed)

A PERIPHERAL DEVICE HAVING A PROGRAMMABLE IDENTIFICATION CONFIGURATION REGISTER

FIELD

[0001] Embodiments of the invention relate to computer devices; and more specifically, to peripheral devices having programmable identification registers.

BACKGROUND

[0002] Computer systems employ a wide variety of peripheral components or input/output (I/O) devices. For example, a typical computer system usually contains a monitor, a keyboard, a mouse, a floppy drive, a network controller, a disk drive or an array of disk drives, and, optionally, a printer. High performance computer systems such as servers have more complex I/O device requirements.

[0003] An example of a host processor of a computer system connected to I/O devices through a component bus is defined by the PCI (peripheral component interconnect) Local Bus Specification, published by the PCI Special Interest Group. During system initialization, the host processor loads a device driver for each PCI device on the PCI bus. A typical PCI device includes multiple configuration registers located within a configuration memory space of each respective PCI device. The configuration registers including identification registers, such as, for example, the vendor ID, device ID or revision register, are read by the device driver and the host system during the initialization or normal operations to identify the PCI device. Typically, the identification registers are hardwired to fixed values during the manufacturing processes of the PCI device and they are not modifiable by the device driver or the operating system (OS) of the host. As a result, a legacy device driver that is looking for

specific identification of a PCI device will not work with a PCI device having different identification information, such as, a different vendor ID or a different device ID, etc.

BRIEF DESCRIPTION OF THE DRAWINGS

[0004] The invention may best be understood by referring to the following description and accompanying drawings that are used to illustrate embodiments of the invention. In the drawings:

[0005] Figure 1 is a block diagram illustrating an exemplary data processing system according to one embodiment.

[0006] Figure 2 is a block diagram illustrating an exemplary configuration containing one or more peripheral devices having a programmable identification registers according to one embodiment.

[0007] Figure 3 is a block diagram illustrating an exemplary peripheral device having one or more configurable identification registers according to one embodiment.

[0008] Figure 4 is a block diagram illustrating an exemplary peripheral device having one or more configurable identification registers according to one embodiment.

[0009] Figure 5 is a block diagram illustrating an exemplary loading sequence of predefined data according to one embodiment.

[0010] Figure 6 is a flow diagram illustrating an exemplary process for programming a peripheral device according to one embodiment.

DETAILED DESCRIPTION

[0011] Methods and apparatuses for programming identifications of a peripheral device are described herein. According to one embodiment, a set of registers in a peripheral device of a data processing system, such as PCI or PCI/PCI bridge device, is implemented to allow a user (e.g., an individual, a reseller, a vendor, or a manufacturer, etc.) to write user definable data (e.g., customized configuration data rather than the default data) into one or more predefined fields of configuration registers of the peripheral device. The configuration registers include at least one identification register, such as a vendor ID, a device ID, and a revision ID register, and the defined data includes the data being written to the at least one identification register (also referred to as ID data). According to one embodiment, when a configuration cycle, such as an ID configuration cycle, is invoked by a data processing system, the ID data is reported instead of the normal default identification data.

[0012] According to one embodiment, during the initialization of the system, such as preset time, data from an external data source (e.g., serial ROM) containing the ID data, as well as the standard configuration information, is transferred to the peripheral device. In one embodiment, during the transfer of the ID data, the host processor (e.g., the microprocessor) is prevented from accessing the configuration registers until the ID data has been transferred and loaded in the peripheral device. A predefined bit pattern included in certain portion of the ROM data would allow the ID data to be written into the configuration registers including at least one identification register (e.g., vendor ID, device ID, and revision ID). The bit pattern may also be used to redirect ID configuration cycles to the registers containing the ID data.

[0013] Accordingly, a configuration cycle invoked to read the device ID then reads the ID data and reports itself to the system software (e.g., device driver or operating system) as the ID data defined by the user. In one embodiment, this ID data persists in the registers until the next initialization event (e.g., preset event), at which time the ID data is reloaded from the

external source (e.g., SROM). The peripheral device may be a regular PCI device or a PCI-X device. Alternatively, the peripheral device may be a PCI bridge device coupling to an upstream primary bus and a downstream secondary bus. Furthermore, the peripheral device may be a PCI Express™ compatible device.

[0014] In the following description, numerous specific details are set forth. However, it is understood that embodiments of the invention may be practiced without these specific details. In other instances, well-known circuits, structures and techniques have not been shown in detail in order not to obscure the understanding of this description.

[0015] Some portions of the detailed descriptions which follow are presented in terms of algorithms and symbolic representations of operations on data bits within a computer memory. These algorithmic descriptions and representations are used by those skilled in the data processing arts to most effectively convey the substance of their work to others skilled in the art. An algorithm is here, and generally, conceived to be a self-consistent sequence of operations leading to a desired result. The operations are those requiring physical manipulations of physical quantities. Usually, though not necessarily, these quantities take the form of electrical or magnetic signals capable of being stored, transferred, combined, compared, and otherwise manipulated. It has proven convenient at times, principally for reasons of common usage, to refer to these signals as bits, values, elements, symbols, characters, terms, numbers, or the like.

[0016] It should be borne in mind, however, that all of these and similar terms are to be associated with the appropriate physical quantities and are merely convenient labels applied to these quantities. Unless specifically stated otherwise as apparent from the following discussion, it is appreciated that throughout the description, discussions utilizing terms such as "processing" or "computing" or "calculating" or "determining" or "displaying" or the like, refer to the action and processes of a computer system, or similar data processing device, that manipulates and transforms data represented as physical (e.g. electronic) quantities within the

computer system's registers and memories into other data similarly represented as physical quantities within the computer system memories or registers or other such information storage, transmission or display devices.

[0017] Embodiments of the present invention also relate to apparatuses for performing the operations described herein. An apparatus may be specially constructed for the required purposes, or it may comprise a general purpose computer selectively activated or reconfigured by a computer program stored in the computer. Such a computer program may be stored in a computer readable storage medium, such as, but is not limited to, any type of disk including floppy disks, optical disks, CD-ROMs, and magnetic-optical disks, read-only memories (ROMs), random access memories (RAMs) such as Dynamic RAM (DRAM), erasable programmable ROMs (EPROMs), electrically erasable programmable ROMs (EEPROMs), magnetic or optical cards, or any type of media suitable for storing electronic instructions, and each of the above storage components is coupled to a computer system bus.

[0018] The algorithms and displays presented herein are not inherently related to any particular computer or other apparatus. Various general purpose systems may be used with programs in accordance with the teachings herein, or it may prove convenient to construct more specialized apparatus to perform the methods. The structure for a variety of these systems will appear from the description below. In addition, embodiments of the present invention are not described with reference to any particular programming language. It will be appreciated that a variety of programming languages may be used to implement the teachings of the embodiments of the invention as described herein.

[0019] A machine-readable medium includes any mechanism for storing or transmitting information in a form readable by a machine (e.g., a computer). For example, a machine-readable medium includes read only memory ("ROM"); random access memory ("RAM"); magnetic disk storage media; optical storage media; flash memory devices; electrical, optical,

acoustical or other form of propagated signals (e.g., carrier waves, infrared signals, digital signals, etc.); etc.

[0020] Figure 1 is a block diagram of an exemplary computer which may be used with an embodiment. For example, exemplary system 100 shown in Figure 1 may perform the process shown in Figure 6. Exemplary system 100 may include one or more peripheral devices having programmable identification registers for identifying the respective peripheral device, similar to those shown in Figures 2-4.

[0021] Note that while Figure 1 illustrates various components of a computer system, it is not intended to represent any particular architecture or manner of interconnecting the components, as such details are not germane to the present invention. It will also be appreciated that network computers, handheld computers, cell phones, and other data processing systems which have fewer components or perhaps more components may also be used with the present invention.

[0022] As shown in Figure 1, the computer system 100, which is a form of a data processing system, includes a bus 102 which is coupled to a microprocessor 103 and a ROM 107, a volatile RAM 105, and a non-volatile memory 106. The microprocessor 103, which may be, for example, a Pentium processor from Intel Corporation or a PowerPC processor from Motorola, Inc., is coupled to cache memory 104 as shown in the example of Figure 1. The bus 102 interconnects these various components together and also interconnects these components 103, 107, 105, and 106 to a display controller and display device 108, as well as to input/output (I/O) devices 110, which may be mice, keyboards, modems, network interfaces, printers, and other devices which are well-known in the art. Typically, the input/output devices 110 are coupled to the system through input/output controllers 109. The volatile RAM 105 is typically implemented as dynamic RAM (DRAM) which requires power continuously in order to refresh or maintain the data in the memory. The non-volatile memory 106 is typically a magnetic hard drive, a magnetic optical drive, an optical drive, or a

DVD RAM or other type of memory system which maintains data even after power is removed from the system. Typically the non-volatile memory will also be a random access memory, although this is not required. While Figure 1 shows that the non-volatile memory is a local device coupled directly to the rest of the components in the data processing system, it will be appreciated that the present invention may utilize a non-volatile memory which is remote from the system, such as a network storage device which is coupled to the data processing system through a network interface such as a modem or Ethernet interface. The bus 102 may include one or more buses connected to each other through various bridges, controllers, and/or adapters, as is well-known in the art. In one embodiment, the I/O controller 109 includes a USB (Universal Serial Bus) adapter for controlling USB peripherals or a PCI controller for controlling PCI devices, which may be included in IO devices 110. In a further embodiment, I/O controller 109 includes an IEEE-1394 controller for controlling IEEE-1394 devices, also known as FireWire devices.

[0023] According to one embodiment, at least one of the PCI devices includes at least one programmable identification register, which may be programmed with predefined data. The predefined data may be stored in a memory within the respective PCI device, such as ROM or serial ROM (SROM). Alternatively, the predefined data may be stored in a system memory, such as ROM 107 or nonvolatile memory 106. According to one embodiment, during an initialization of system 100, such as booting, resetting, or bus re-enumeration of system 100, a configuration cycle, such as an ID configuration cycle, is invoked. During the configuration cycle, the predefined data for a peripheral device (e.g., PCI device) is read from a memory (e.g., SROM) and is used to program one or more configuration registers, including at least one identification register, such as, for example, vendor ID, device ID, and revision ID registers, of the respective peripheral device.

[0024] The configuration cycle may be carried out by system software, such as system software embedded within the chipset or processor, such as processor 103. Alternatively, the

configuration cycle may be carried out by the BIOS (basic input and output system) or a device driver of an operating system (OS). The operating system may be a Windows operating system from Microsoft Corporation of Redmond, Washington or a Mac OS from Apple Computer of Cupertino, California. Alternatively, the operating system may be a Linux or Unix operating system. Other operating systems, such as real-time embedded operating systems may be utilized.

[0025] According to one embodiment, the predefined data may be loaded in a sequence similar to the exemplary loading sequence 500 of Figure 5. One or more bit patterns of the predefined data may be used to indicate whether the predefined data should be loaded, or alternatively, default data should be used. Once the predefined data has been loaded in the configuration registers of the peripheral device, the system may perform a verification of the configuration registers. For example, the programmed configuration registers may be read and compared with the predefined data stored in the memory to verify whether the respective configuration registers have been successfully configured. Once the configuration registers of the peripheral device is verified, the peripheral device may be enabled. Thereafter, the enabled peripheral device may report its predefined identity, rather than a default identity, to a system query. In one embodiment, once the peripheral device is enabled, the identification registers, such as vendor ID, device ID, and revision ID registers, may be configured as read-only registers, such that the enabled peripheral device operates in compliance with the associated specification, such as PCI specification.

[0026] With the programmable identification, according to one embodiment, a user may update a peripheral device or subsystem in implementations where the system software or device driver has been written to require a specific response to the ID configuration cycle (e.g., a vendor ID or a device ID cycle). This feature is typically useful where a specific device is no longer available and it is not possible to modify the system software or device drivers. By allowing the new device to "masquerade" as the previous device. For example, a

customer could develop a modern functionality equivalent subsystem that identifies itself as a legacy device even though the legacy device has been obsolete by the respective manufacturer. This could be accomplished without requiring modification to the operating system or device drivers. This will also allow the customers to extend the useful life of a system design beyond the end of life (EOL) of individual components.

[0027] In addition, according to one embodiment, a user may control the allowable devices permitted to be inserted into a system by requiring a specific identification (e.g., a vendor ID or a device ID, etc.) to be found. In the real-time mission critical systems, it would be possible to control the devices installed in the system by requiring the identification data fields of configuration registers to match the customer definable data. This could increase the overall reliability of a computing system by assuring only the validated devices are used in the system. The user may also use this feature in "branding" an option card to reflect the user's corporation, similar to re-labeling the peripheral device to match the vendor ID or device ID of the corporation.

[0028] Furthermore, a customer may implement a subsystem in a computing system that would "masquerade" as a standard device, which may be undetectable by normal user interaction. This could be used as a security device in monitoring system I/O activities or by activating a non-visible subsystem that could be triggered under the controlled circumstances. For example, a PCI device normally providing communication functionality could also contain a monitoring subsystem although it would report itself as a standard communication device.

[0029] Figure 2 is a block diagram illustrating an exemplary configuration including one or more peripheral devices having a programmable identification registers in accordance with one embodiment. In this embodiment, exemplary system 200 includes a plug-in card 201, also referred to as an option card, which may be plugged into host system slot 203, such as a primary PCI bus, of system board 202 (e.g., motherboard of exemplary system 100 of Figure

1). In one embodiment, option card 201 includes one or more peripheral devices, such as PCI/PCI-X bridge 205, and PCI/PCI-X devices 206-209. The bridge device 205 provides an interface between a primary bus 203 and a secondary bus 204 for coupling peripheral devices 206-209. Note that devices 206-209 are not limited to PCI/PCI-X devices. It will be appreciated that devices 206-209 may be implemented as other peripheral devices, such as, for example, PCI Express™ compatible devices.

[0030] According to one embodiment, bridge device 205 includes one or more programmable or configurable identification registers, such as, for example, vendor ID, device ID, and revision ID registers, which may be programmable during an initialization phase of system 200, such as preset time. During the initialization period of system 200, bridge device 205 detects a signal, such as a reset signal, received from the main system board 202, such as a chipset or microprocessor (e.g., processor 103 of system 100 shown in Figure 1), indicating that an initialization occurs. In response to the initialization, predefined data for the configuration registers of device 205 is loaded into the configuration registers including at least identification register, such as vendor ID, device ID, and revision ID registers.

[0031] In one embodiment, during the transfer of the ID data, the host processor (e.g., the microprocessor) is prevented from accessing the configuration registers until the ID data has been transferred and loaded in the peripheral device. For example, when the peripheral device is a PCI device, the host processor may be prevented from accessing the configuration registers until a retry sequence has been completed. In this embodiment, the host processor will be substantially guaranteed that the host will not observe the default values rather than the newly programmed values. Note that the programming of the ID registers may occur without the retry behavior.

[0032] The predefined data may be stored in a memory, such as SROM 206, coupled to the bridge device 205. SROM 306 may be located within device 205. Alternatively, SROM 206 may be located outside of device 205, as shown in Figure 2. According to one

embodiment, SROM 206 may be shared between bridge device 205 and other peripheral devices, such as devices 207-209. According to one embodiment, devices 207-209 may also include one or more programmable identification registers (e.g., vendor ID, device ID, and revision ID registers, etc.). Each of the devices 207-209 may individually include a memory, such as SROM, to store the respective user predefine data that can be loaded into the respective configuration registers including at least one identification register. Alternatively, devices 207-209 may share SROM 206 with bridge device 205 to store their respective predefined data.

[0033] According to one embodiment, the predefined data may be stored in SROM 206 during manufacturing of the option card 201 or bridge device 205. Alternatively, the predefined data may be loaded into SROM 206 through a utility, such as a flash utility, that reads the predefined data from a file, such as text or script file, and writes the data into the SROM 206. An exemplary predefined data file is shown in the Appendix A of the present application.

[0034] According to one embodiment, the predefined data may be loaded into SROM 206 per application basis. For example, when option card 201 is about to be plugged into system slot 203 of system 200, a user may discover that the system software or a device driver of system 200 is looking for particular vendor ID, device ID, or revision ID of a peripheral device represented by option card 201. Accordingly, in order allow the option card 201 to be recognized by the system software or the device driver, the user may modify the predefined data file, similar to the one shown in Appendix A, to match the identification (e.g., vendor ID, device ID, or revision ID) the system software or the device driver is looking for, and load the modified predefined data file into SROM 206.

[0035] According to another embodiment, a use predefined data file may be provided by a vendor or manufacturer of option card 201 and the data file may be downloaded over a

network from a remote facility via a network interface, such as network interface 110 of system 100 shown in Figure 1. Other configurations may exist.

[0036] Thereafter, in one embodiment, system 200 may have to be rebooted to enter another initialization period. During the initialization phase (e.g., preset time), the bridge device 205 loads the predefined data from SROM 206 into its configuration registers, including at least one identification register (e.g., vendor ID, device ID, and revision ID registers). The predefined data may be loaded according to exemplary loading sequence 500 shown in Figure 5. After the predefined data has been loaded into the configuration registers of device 205, the bridge device 205 may perform a verification process to verify whether the predefined data has been successfully loaded into the configuration registers. In one embodiment, the bridge device 205 may perform read operations from the configuration registers and compare the data with the data stored in SROM 206. Other mechanisms may exist.

[0037] Once the bridge device 205 has been successfully programmed using the predefined data from SROM 206, the bridge device 205 is enabled. In response to a query, such as bus re-enumeration or a query from the respective device driver, the enabled device 205 may respond using the programmed predefined data to identify itself as the device represented by the predefined data. As a result, the system software or the device driver recognizes the programmed device 205 without modifying the system software or the device driver.

[0038] Figure 3 is a block diagram illustrating an exemplary peripheral bridge device having one or more programmable identification registers according to one embodiment. Exemplary bridge device 300 may be used as peripheral bridge device 205 of option card 201 shown in Figure 2. In one embodiment, exemplary bridge device 300 includes a processor to perform one or more peripheral functions, one or more programmable configuration registers accessible by the processor, the one or more programmable configuration registers including

at least one identification register for identifying the peripheral device, and a memory coupled to the processor to store predefined data, the predefined data being used to program the one or more programmable configuration registers including the at least one identification register in response to a configuration cycle of the peripheral device.

[0039] Referring to Figure 3, peripheral device 300 includes a primary bus interface 301 and a secondary bus interface 302 to connect primary bus 303 and secondary bus 304 respectively. Primary bus 303 and secondary bus 304 may be PCI/PCI-X or PCI Express™ compatible buses and peripheral device 300 may be a PCI/PCI bridge or a PCI Express bridge. Primary bus interface 301 generally connects to an upstream bus 303 via an upstream data path 309. Secondary bus interface 302 generally attaches to a downstream bus 304 via downstream data path 308. Primary bus interface 303 may be used as host primary bus 203 and secondary bus 304 may be used as secondary PCI bus 204 shown in Figure 2.

[0040] According to one embodiment, peripheral device 300 includes one or more programmable configuration registers, which includes at least one programmable identification register, such as, for example, vendor ID, device ID, and revision ID, etc. During an initialization process of peripheral device 300, which may be detected and processed by reset processor 307, predefined data may be read from a memory, such as SROM 305, and may be programmed into one or more configuration registers of peripheral device 300, including at least one identification register, instead of default configuration data. The predefined data may be loaded according to a loading sequence, similar to loading sequence 500 of Figure 5. The predefined data may be loaded and stored in SROM 305 via a utility based on an input file defining the user data, similar to the one shown in Appendix A.

[0041] According to one embodiment, when peripheral device 300 detects that the device is in a reset state (e.g., P_RST# goes inactive, which may be detected by reset processor 307), a peripheral device 300 automatically executes a read from serial EEPROM (e.g., SROM 305). If the predetermined bit pattern is detected during the ROM load, the appropriate fields,

including at least one identification field, of the configuration registers of peripheral device 300 are loaded with the predefined data read from SROM 305. In one embodiment, the bit pattern indicating that the predefined data is loaded includes the most significant bit (MSB) of the first byte having a logical one value, as shown in first byte 501 of load sequence 500 in Figure 5. At the completion of the ROM load, a reset line (e.g., S_RST#) is de-asserted. Thereafter, peripheral device 300 is enabled and responds to ID configuration cycles by reporting an identification represented by the user provided data. In one embodiment, this behavior persists without requiring additional accesses to the ROM (e.g., SROM 305) or imposing additional latency until the next initialization cycle (e.g., P_RST# event), which reloads the ROM and begins the cycle again.

[0042] In one embodiment, during loading of the ID data, the host processor (e.g., the microprocessor) is prevented from accessing the configuration registers until the ID data has been completely transferred and loaded in the peripheral device. For example, when the peripheral device is a PCI device, the host processor may be prevented from accessing the configuration registers until a retry sequence has been completed. In this embodiment, the host processor will be substantially guaranteed that the host will not observe the default values rather than the newly programmed values. Note that the programming of the ID registers may occur without the retry behavior as described above.

[0043] Peripheral device 300 may further include other components, such as primary and secondary clock generators for asynchronous operations between primary bus 303 and secondary bus 304. Peripheral device 300 may further includes a hot swap control unit 310, GPIO (general purpose input and output) unit 306, secondary arbiter 312, and configuration unit 314. Hot swap control unit 310 allows the peripheral device 300 to be inserted or removed from primary bus 303 without requiring the system to be shut down. Secondary arbiter 312 handles arbitration to support multiple devices coupled to the secondary bus 304, such as, for example, peripheral devices 207-209 coupled to the secondary bus 204 shown in

Figure 2. Other components apparent to those with ordinary skill in the art, such as control logic or multiplexers, etc., may be included.

[0044] According to one embodiment, SROM 305 may be a 512-byte serial EEPROM that is used for loading the configuration space registers settings of a peripheral device (e.g., peripheral device 300) prior to host system discovery and initialization of the device. For example, SROM 305 interface may be compatible with industry standard 512-byte Microwire™ serial ROM, such as 93LC66A 512x8 serial EEPROM or equivalent, available from Microchip Technology, Inc. However, SROM 305 is not limited those discussed above. Other memory or configurations may be utilized.

[0045] In one embodiment, the SROM interface may include at least one of the interface signals:

Name	Type	Description
SR_CS	O	SROM chip select
SR_CLK	O	SROM clock
SR_DI	O	SROM data in
SR_DO	I	SROM data out

Figure 4 is a block diagram illustrating an exemplary configuration between a peripheral device and an SROM according to one embodiment. In one embodiment, exemplary configuration 400 includes a peripheral device 401 coupled with an SROM 402 via one or more SROM interface using at least one of the above interface signals.

[0046] Figure 5 is a block diagram illustrating an exemplary loading sequence of an SROM image according to one embodiment. According to one embodiment, SROM image 500 may be present, but not desired to be loaded, which is indicated by one or more bit patterns. In a particular embodiment, if the SROM is present, but a register preload is not desired, bits [7:6] of the first byte 501 (e.g., the first two bits read) should be read as any value except the preload enable value, such as value of 10b (binary).

[0047] Figure 6 is a flow diagram illustrating an exemplary process for programming one or more configuration registers including at least one identification register, according to one embodiment. Exemplary process 600 may be performed by a processing logic that may comprise hardware (circuitry, dedicated logic, etc.), software (such as is run on a general purpose computer system or a dedicated machine), or a combination of both. In one embodiment, exemplary process 600 includes programming, based on predefined data, one or more fields of configuration registers of a peripheral device in response to a configuration cycle of a data processing system, where the one or more fields of the configuration registers include at least one identification register for identifying the peripheral device. In one embodiment, exemplary process 600 may be performed by system software, such as, for example, BIOS, kernel of an operating system, or a device driver.

[0048] Referring to Figure 6, at block 601, a data processing system start to perform an initialization process, such as power up or reboot process, the data processing system including one or more peripheral devices, such as PCI, PCI-X, or PCI Express™ compatible devices. At block 602, for each peripheral device, processing logic determines whether the respective peripheral device needs to be configured using predefined data. In one embodiment, the determination is performed based on whether there is a memory, such as ROM or SROM, associated with the peripheral device. Alternatively, the determination may be based on whether there is predefined data stored in the memory. Furthermore, even if there is predefined data stored in the memory, the determination is based on whether the predefined data is enabled, which may be based on one or more bit patterns of the data. In particular embodiment, the most significant bit of the first byte of the data image indicates that the data image is enabled.

[0049] If it is determined that the respective peripheral device needs to be programmed using predefined data, at block 603, processing logic retrieves the predefined data from the memory (e.g., SROM) and at block 604, the retrieved data is used to program one or more

configuration registers of the peripheral device, including at least one of the identification registers, such as vendor ID, device ID, and revision ID, etc. Once the peripheral device has been successfully programmed, at block 605, the peripheral device is enabled for use in the data processing system, including, but not limited to, allocating memory and loading corresponding device driver or drivers for the peripheral device.

[0050] If it is determined that the respective peripheral device does not need to be programmed using predefined data, at block 606, a default data is loaded with the peripheral device. Other operations apparent to those with ordinary skills in the art may be included.

[0051] Thus, methods and apparatuses for programming identifications of a peripheral device have been described. In the foregoing specification, the invention has been described with reference to specific exemplary embodiments thereof. It will be evident that various modifications may be made thereto without departing from the broader spirit and scope of the invention as set forth in the following claims. The specification and drawings are, accordingly, to be regarded in an illustrative sense rather than a restrictive sense.

Appendix A

```
; This file will create an SROM image
; This loads the registers with their reset values

[
; preload enable (sets bit 7 for preload enable)
:0      80
:1      00
:2      00
:3      00
; Primary Class Code
:4      00
:5      80
:6      06
; Subvendor IDs
:7      46
:8      00
:9      11
:A      10
; Primary Min GNT, Max LAT
:B      00
:C      00
; Secondary Class Code
:D      00
:E      80
:F      06
; Secondary Min GNT,Max LAT
:10     00
:11     00
;Downstream Mem 0 -CSRs only (Set a 4K window size)
:12     00
:13     F0
:14     FF
:15     FF
; Downstream Mem I or 1/0 (Set 256 byte 1/0 window size)
:16     01
:17     FF
:18     FF
:19     FF
; Downstream Mem 2 (Set 8MB memory window size)
:1A     08
:1B     00
:1C     80
```

```
:1D     FF
; Downstream Mem 3 (Set 8MB memory window size)
:1E     08
:1F     00
:20     80
:21     FF
; Downstream Mem 3 Upper 32
:22     00
:23     00
:24     00
:25     00
; Expansion ROM (Set 1MB expansion ROM size)
:26     01
:27     F0
; Upstream Mem 0 or 1/0 (Set 256 byte 1/0 window size)
:28     01
:29     FF
:2A     FF
:2B     FF
; Upstream Mem I (Set 8MB memory window size)
:2C     08
:2D     00
:2E     80
:2F     FF
; Chip Control 0
:30     00
; clear lockout bit
:31     00
; Chip Control I
:32     00
; LUT disable
:33     00
; Arbiter control
:34     00
:35     02
; System error disable
:36     00
:37     00
; Power management
:38     00
:39     00
:3A     00
:38     00
```

```
:3C     00
:30     00
:3E     00
:3F     00
:40     00
:41     00
:42     00
]
```